Forward stochastic reachability using Fourier transforms

This example will demonstrate the use of SReachTools in forward stochastic reachability analysis for stochastic continuous-state discrete-time linear time-invariant (LTI) systems.

Specifically, we will discuss how SReachTools uses Fourier transforms to efficiently compute

1. **Forward stochastic reach set**: The support of the random vector describing the state.
2. **Forward stochastic reach probability density**: The probability density function associated with the random vector describing the state

at a future time of interest.

Our approach is grid-free and recursion-free resulting in highly scalable solutions, especially for Gaussian-perturbed LTI systems. We will consider the case where the initial state is a known deterministic point in the state space, and the case where the initial state is a random vector.

Notes about this Live Script:

1. **MATLAB dependencies**: This Live Script uses MATLAB's Statistics and Machine Learning Toolbox and Control System Toolbox.
2. **External dependencies**: This Live Script uses Multi-Parameteric Toolbox (MPT).
3. We will also Genz's algorithm (included in helperFunctions of SReachTools) to evaluate integrals of a Gaussian density over a polytope.
4. Make sure that `srtinit` is run before running this script.

Problem formulation: Spacecraft motion via CWH dynamics

We consider both the spacecrafts, referred to as the deputy spacecraft and the chief spacecraft, to be in the same circular orbit. In this example, we will consider the forward stochastic reachability analysis of the deputy.

Dynamics model for the deputy relative to the chief spacecraft

The relative planar dynamics of the deputy with respect to the chief are described by the Clohessy-Wiltshire-Hill (CWH) equations,

$$\ddot{x} - 3\omega x - 2\omega \dot{y} = \frac{F_x}{m_d}$$

$$\ddot{y} + 2\omega \dot{x} = \frac{F_y}{m_d}$$

where the position of the deputy relative to the chief is $x, y \in \mathbf{R}$, $\omega = \sqrt{\frac{\mu}{R_0^3}}$ is the orbital frequency, μ is the gravitational constant, and R_0 is the orbital radius of the chief spacecraft. We define the state as $\overline{x} = [x\ \ y\ \ \dot{x}\ \ \dot{y}]^\top \in \mathbf{R}^4$ which is the position and velocity of the deputy relative to the chief along x- and y- axes, and the input as $\overline{u} = [F_x\ \ F_y]^\top \in \mathscr{U} \subset \mathbf{R}^2$.

We will discretize the CWH dynamics in time, via zero-order hold, to obtain the discrete-time linear time-invariant system and add a Gaussian disturbance to account for the modeling uncertainties and the disturbance forces,

$$\overline{x}_{k+1} = A\overline{x}_k + B\overline{u}_k + \overline{w}_k$$

with $\overline{w}_k \in \mathbf{R}^4$ as an IID Gaussian zero-mean random process with a known covariance matrix $\Sigma_{\overline{w}}$.

SReachTools directly allows us to create a `LtiSystem` object with these dynamics. We will set the input space to be unbounded.

```
umax=Inf;
mean_disturbance = zeros(4,1);
covariance_disturbance = diag([1e-4, 1e-4, 5e-8, 5e-8]);
% Define the CWH (planar) dynamics of the deputy spacecraft relative to the chief spac
sys_CWH = getCwhLtiSystem(4,...
                          Polyhedron('lb', -umax*ones(2,1),...
                                     'ub',  umax*ones(2,1)),...
                          StochasticDisturbance('Gaussian',...
                                                mean_disturbance,...
                                                covariance_disturbance));
disp(sys_CWH);
```

```
Linear time invariant system with 4 states, 2 inputs, and 4 disturbances.
```

Creating a LtiSystem object describing the dynamics of the deputy under the action of a linear feedback law

We will define a `LtiSystem` object to describe the dynamics when $\overline{u}_k = -K\overline{x}_k$ for some $K \in \mathbf{R}^{4\times 2}$. We will compute K using LQR theory with $Q = 0.01 I_4$ and $R = I_2$, i.e., $\overline{u}_k = -K\overline{x}_k$ will regulate the deputy spacecraft towards the origin.

```
% Create a discrete-time LQR controller that regulates the deputy to the origin
K = lqr(ss(sys_CWH.state_mat,sys_CWH.input_mat,[],[],-1),0.01*eye(4),eye(2));
% Reuse the system definition in sys_CWH with appropriately defined state matrix
closed_loop_state_mat = sys_CWH.state_mat - sys_CWH.input_mat*K;
sys = LtiSystem('StateMatrix', closed_loop_state_mat,...
                'DisturbanceMatrix', sys_CWH.dist_mat,...
                'Disturbance', sys_CWH.dist);
disp(sys);
```

```
Linear time invariant system with 4 states, 0 inputs, and 4 disturbances.
```

Problem 1: What is the probability that the deputy rendezvous with the chief satellite at some future time of interest?

Since the chief is located at the origin in this coordinate frame (`sys` describes the relative dynamics of the deputy), we define the target set to be a small box centered at the origin (`target_set` is a box axis-aligned with side 0.2). We are interested in the probability that the deputy will meet the chief at `target_time` time steps in future.

```
target_time = 23;                                           % Time of interest
target_set = Polyhedron('lb',-0.05 * ones(4,1),...
                        'ub', 0.05 * ones(4,1));             % Target set definition
desired_accuracy = 1e-2;
```

Problem 1a: Fixed initial state

```
initial_state = [-10;
                  10;
                  0;
                  0];                                        % Initial state definition
```

1. Compute the mean and the covariance of the forward stochastic reach probability density of the state at time `target_time starting from this fixed initial state.`

```
[mean_x, cov_x] = getFSRPDMeanCov(sys,initial_state, target_time);
disp(mean_x);
```

```
   -0.0056
    0.0229
   -0.0001
    0.0003
```

```
disp(cov_x);
```

```
   1.0e-03 *

    0.4935   -0.0000   -0.0026   -0.0003
   -0.0000    0.4937    0.0003   -0.0026
   -0.0026    0.0003    0.0001    0.0000
   -0.0003   -0.0026    0.0000    0.0001
```

2. Compute the probability of reaching a target set at a specified target time

```
% Integrate the FSRPD at time target_time over the target_set
prob = getProbReachSet(sys,...
                       initial_state,...
                       target_set,...
                       target_time,...
                       desired_accuracy);
fprintf('Probability of x_{target_time} lying in target_set: %1.4f\n',prob);
```

```
Probability of x_{target_time} lying in target_set: 0.8600
```

3. Validate this probability via Monte-Carlo simulations

```
n_mcarlo_sims = 1e5;
% This function returns the concatenated state vector stacked columnwise
concat_state_realization = generateMonteCarloSims(...
                                                  n_mcarlo_sims,...
                                                  sys,...
                                                  initial_state,...
                                                  target_time);
% Extract the location of the deputy at target_time
end_locations = concat_state_realization(end-sys.state_dim +1 : end,:);
% Check if the location is within the target_set or not
mcarlo_result = target_set.contains(end_locations);
if abs(sum(mcarlo_result)/n_mcarlo_sims - prob)>desired_accuracy
    error(sprintf('Failed sanity check. Error of %1.3e', abs(sum(mcarlo_result)/n_mcar
end
fprintf('Monte-Carlo simulation using %1.0e particles: %1.3f\n',...
        n_mcarlo_sims,...
        sum(mcarlo_result)/n_mcarlo_sims);
```

```
Monte-Carlo simulation using 1e+05 particles: 0.861
```

Problem 1b: Initial state is a Gaussian random vector

```
initial_state_rv = RandomVector('Gaussian',...
                             initial_state,...
                             0.1*eye(4));         % Initial state definition
```

1. Compute the mean and the covariance of the forward stochastic reach probability density of the state at time `target_time when the initial state is stochastic.`

```
[mean_x, cov_x] = getFSRPDMeanCov(sys,initial_state_rv, target_time);
disp(mean_x);
```

```
   -0.0056
    0.0229
   -0.0001
    0.0003
```

```
disp(cov_x);
```

```
   1.0e-03 *

    0.9818   -0.0100   -0.0263   -0.0007
   -0.0100    0.9350    0.0010   -0.0254
   -0.0263    0.0010    0.0013    0.0000
   -0.0007   -0.0254    0.0000    0.0013
```

2. Compute the probability of reaching a target set at a specified target time

```
% Integrate the FSRPD at time target_time over the target_set
prob = getProbReachSet(sys,...
                       initial_state_rv,...
```

```
                          target_set,...
                          target_time,...
                          desired_accuracy);
fprintf('Probability of x_{target_time} lying in target_set: %1.4f\n',prob);
```

```
Probability of x_{target_time} lying in target_set: 0.7100
```

Notice how the probability of success is lower due to a random initial state.

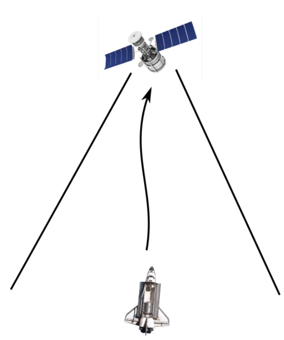
3. Validate this reach probability via Monte-Carlo simulations

```
n_mcarlo_sims = 1e5;
% This function returns the concatenated state vector stacked columnwise
concat_state_realization = generateMonteCarloSims(...
                                               n_mcarlo_sims,...
                                               sys,...
                                               initial_state_rv,...
                                               target_time);
% Extract the location of the deputy at target_time
end_locations = concat_state_realization(end-sys.state_dim +1 : end,:);
% Check if the location is within the target_set or not
mcarlo_result = target_set.contains(end_locations);
if abs(sum(mcarlo_result)/n_mcarlo_sims - prob)>desired_accuracy
    error(sprintf('Failed sanity check. Error of %1.3e', abs(sum(mcarlo_result)/n_mcar
end
fprintf('Monte-Carlo simulation using %1.0e particles: %1.3f\n',...
        n_mcarlo_sims,...
        sum(mcarlo_result)/n_mcarlo_sims);
```

```
Monte-Carlo simulation using 1e+05 particles: 0.712
```

Problem 2: What is the probability that the deputy rendezvous with the chief satellite at some future time of interest while staying within a line-of-sight cone?

Since the chief is located at the origin in this coordinate frame (`sys` describes the relative dynamics of the deputy), we define the target set to be a small box centered at the origin (`target_set` is a box axis-aligned with side 0.2). We are interested in the probability that the deputy will meet the chief at `target_time` time steps in future. **Additionally**, we desire that the deputy satellite stays within a line-of-sight cone for accurate sensing.



```
target_time = 10;                                                  % Time of interest
```

```
target_set = Polyhedron('lb',-0.05 * ones(4,1),...
                        'ub', 0.05 * ones(4,1));         % Target set definition
%% Safe set definition --- LoS cone |x|<=y and y\in[0,ymax] and |vx|<=vxmax and |vy|<=
ymax=10;
vxmax=0.5;
vymax=0.5;
A_safe_set = [1, 1, 0, 0;
             -1, 1, 0, 0;
              0, -1, 0, 0;
              0, 0, 1,0;
              0, 0,-1,0;
              0, 0, 0,1;
              0, 0, 0,-1];
b_safe_set = [0;
              0;
              ymax;
              vxmax;
              vxmax;
              vymax;
              vymax];
safe_set = Polyhedron(A_safe_set, b_safe_set);
% Create a target tube
target_tube = TargetTube('reach-avoid', safe_set, target_set, target_time);
```

Problem 2a: Fixed initial state

```
initial_state = [0;
                 -1;
                 0;
                 0];                                         % Initial state definition
```

1. Compute the mean and the covariance of the forward stochastic reach probability density of the state at time `target_time starting from this fixed initial state.`

```
[~, mean_X, ~] = getHmatMeanCovForXSansInput(sys, initial_state, target_time);
mean_X_trajectory = reshape(mean_X,4,[]);
disp(mean_X_trajectory);
```

```
   -0.0034   -0.0098   -0.0145   -0.0159   -0.0142   -0.0102   -0.0051    0.0001    0.0046    0.0079
   -0.9486   -0.8201   -0.6587   -0.4943   -0.3451   -0.2204   -0.1232   -0.0526   -0.0051    0.0237
   -0.0003   -0.0003   -0.0002    0.0000    0.0002    0.0002    0.0003    0.0002    0.0002    0.0001
    0.0051    0.0077    0.0084    0.0080    0.0069    0.0056    0.0042    0.0029    0.0018    0.0010
```

2. Compute the probability of reaching a target set at a specified target time **while staying within a safe set**

```
% Integrate the FSRPD at time target_time over the target_set
prob = getProbReachTargetTube(sys,...
                              initial_state,...
                              target_tube,...
                              desired_accuracy);
```

```
fprintf('Probability of x_{target_time} lying in target_set while staying inside line-
```

```
Probability of x_{target_time} lying in target_set while staying inside line-of-sight cone: 0.3100
```

3. Validate this reach probability via Monte-Carlo simulations

```
n_mcarlo_sims = 1e5;
% This function returns the concatenated state vector stacked columnwise
concat_state_realization = generateMonteCarloSims(...
                                               n_mcarlo_sims,...
                                               sys,...
                                               initial_state,...
                                               target_time);
% Check if the location is within the target_set or not
mcarlo_result = target_tube.contains([repmat(initial_state,1,n_mcarlo_sims);
                                      concat_state_realization]);
if abs(sum(mcarlo_result)/n_mcarlo_sims - prob)>desired_accuracy
    error(sprintf('Failed sanity check. Error of %1.3e', abs(sum(mcarlo_result)/n_mcar
end
fprintf('Monte-Carlo simulation using %1.0e particles: %1.3f\n',...
        n_mcarlo_sims,...
        sum(mcarlo_result)/n_mcarlo_sims);
```

```
Monte-Carlo simulation using 1e+05 particles: 0.312
```

Problem 2b: Initial state is a Gaussian random vector

```
initial_state_rv = RandomVector('Gaussian',...
                                initial_state,...
                                0.001*eye(4));         % Initial state definition
```

1. Compute the mean and the covariance of the forward stochastic reach probability density of the state at time `target_time starting from this fixed initial state.`

```
[~, mean_X, ~] = getHmatMeanCovForXSansInput(sys, initial_state_rv, target_time);
mean_X_trajectory = reshape(mean_X,4,[]);
disp(mean_X_trajectory);
```

```
   -0.0034   -0.0098   -0.0145   -0.0159   -0.0142   -0.0102   -0.0051    0.0001    0.0046    0.0079
   -0.9486   -0.8201   -0.6587   -0.4943   -0.3451   -0.2204   -0.1232   -0.0526   -0.0051    0.0237
   -0.0003   -0.0003   -0.0002    0.0000    0.0002    0.0002    0.0003    0.0002    0.0002    0.0001
    0.0051    0.0077    0.0084    0.0080    0.0069    0.0056    0.0042    0.0029    0.0018    0.0010
```

Note that the mean remains the same as Problem 2a.

2. Compute the probability of reaching a target set at a specified target time **while staying within a safe set**

```
% Integrate the FSRPD at time target_time over the target_set
prob = getProbReachTargetTube(sys,...
```

```
                                initial_state_rv,...
                                target_tube,...
                                desired_accuracy);
fprintf('Probability of x_{target_time} lying in target_set while staying inside line-
```

```
Probability of x_{target_time} lying in target_set while staying inside line-of-sight cone: 0.0600
```

However, the probability decreases drastically since the initial state is now random.

3. Validate this reach probability via Monte-Carlo simulations

```
n_mcarlo_sims = 1e5;
% This function returns the concatenated state vector stacked columnwise
concat_state_realization = generateMonteCarloSims(...
                                               n_mcarlo_sims,...
                                               sys,...
                                               initial_state_rv,...
                                               target_time);
% Check if the location is within the target_set or not
mcarlo_result = target_tube.contains([repmat(initial_state,1,n_mcarlo_sims);
                                      concat_state_realization]);
if abs(sum(mcarlo_result)/n_mcarlo_sims - prob)>desired_accuracy
    error(sprintf('Failed sanity check. Error of %1.3e', abs(sum(mcarlo_result)/n_mcar
end
fprintf('Monte-Carlo simulation using %1.0e particles: %1.3f\n',...
        n_mcarlo_sims,...
        sum(mcarlo_result)/n_mcarlo_sims);
```

```
Monte-Carlo simulation using 1e+05 particles: 0.065
```